

Gedae: Auto Coding to a Virtual Machine

William I. Lundgren, Kerry B. Barnes, and James W. Steed

Gedae, Inc.

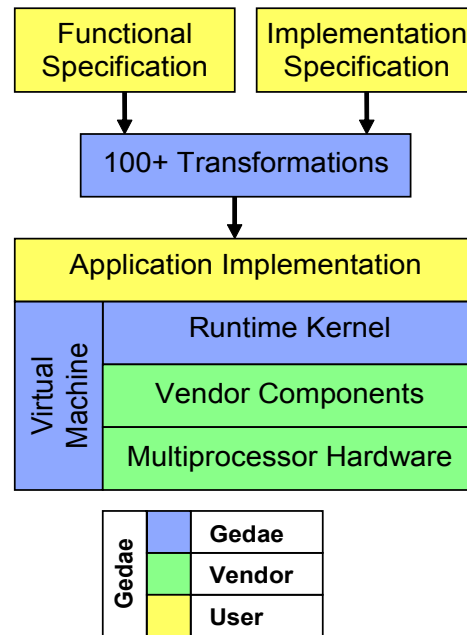
Phone: 856-231-4458

Email Addresses: {bill.lundgren, kerry.barnes, [jim](mailto:jim@gedae.com)}@gedae.com

Gedae is an integrated application development environment. It has been under development since 1987 – though the concepts involved are rooted in much earlier work done in the areas of data flow and hardware simulation. In Gedae we have developed a language for describing an architecture-independent functional specification, a virtual machine on which the application runs, and transformations that create an efficient implementation of the application that runs on the virtual machine. In this paper we discuss three topics – the language, the virtual machine and the transformations.

The language was developed with two requirements – any functionality must be easily expressible, and the language must be transformable into an efficient implementation on the virtual machine. The Gedae Language consists of both the Gedae Primitive Language and the Gedae Graph Language. Much of the expressiveness is in the primitive description language. The language has over 50 expression features to define the behavior of functional ports. Port data flow requirements can be specified either prior to runtime (static) or at runtime (dynamic). Ports can add segment boundary markers on the data flow streams, thereby breaking the stream into independent data sets. Exclusive families of ports can send data down one branch or another to implement mode changes while maintaining coherent state vectors used by all the modes. Primitives can maintain their own local state variables and provide methods for execution, startup, termination and handling the beginning and ending of segment processing. The Gedae Graph Language allows the hierarchical development of graphs consisting of primitives, parameters and other Gedae graphs. The graph language can describe families of these entities to allow parameterized expression of parallelism. The resulting language permits

The Structure of Gedae



direct expression of signal and data processing algorithms, distribution for providing load balancing and fault tolerance, and application (or software, or mode) control.

To achieve efficiency, the language and virtual machine were codesigned. The virtual machine contains a runtime kernel that executes components generated by the transformations. For example, the static scheduler executes predetermined execution sequences based on static data flow ports, and the dynamic scheduler executes groups of static schedules that interface through dynamic data flow ports. The virtual machine manages the segment processing and controls the efficient and timely transfer of distributed state vectors between processors. The virtual machine also allows for vendor specific optimizations of processing, such as, setting data transfer parameters. A thin layer

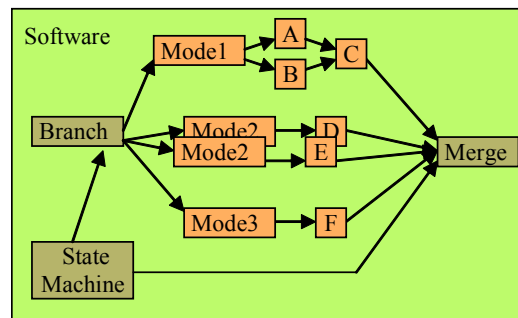
Report Documentation Page				Form Approved OMB No. 0704-0188	
Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.					
1. REPORT DATE 01 FEB 2005		2. REPORT TYPE N/A		3. DATES COVERED -	
4. TITLE AND SUBTITLE Gedae: Auto Coding to a Virtual Machine				5a. CONTRACT NUMBER	
				5b. GRANT NUMBER	
				5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S)				5d. PROJECT NUMBER	
				5e. TASK NUMBER	
				5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Gedae, Inc.				8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)				10. SPONSOR/MONITOR'S ACRONYM(S)	
				11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release, distribution unlimited					
13. SUPPLEMENTARY NOTES See also ADM00001742, HPEC-7 Volume 1, Proceedings of the Eighth Annual High Performance Embedded Computing (HPEC) Workshops, 28-30 September 2004 Volume 1., The original document contains color images.					
14. ABSTRACT					
15. SUBJECT TERMS					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT UU	18. NUMBER OF PAGES 32	19a. NAME OF RESPONSIBLE PERSON
a. REPORT unclassified	b. ABSTRACT unclassified	c. THIS PAGE unclassified			

over the vendor-provide vector processing libraries allows primitives to execute efficiently.

One of the unique features of Gedae is the visibility of the implementation and the execution it provides. This visibility is possible because the language, the transformations and the virtual machine are all part of Gedae. The visibility allows the generation of detailed execution timelines and the symbolic viewing of any memory in the system. Primitive execution, queue state and data transfers between processors can be dynamically viewed when the application is running.

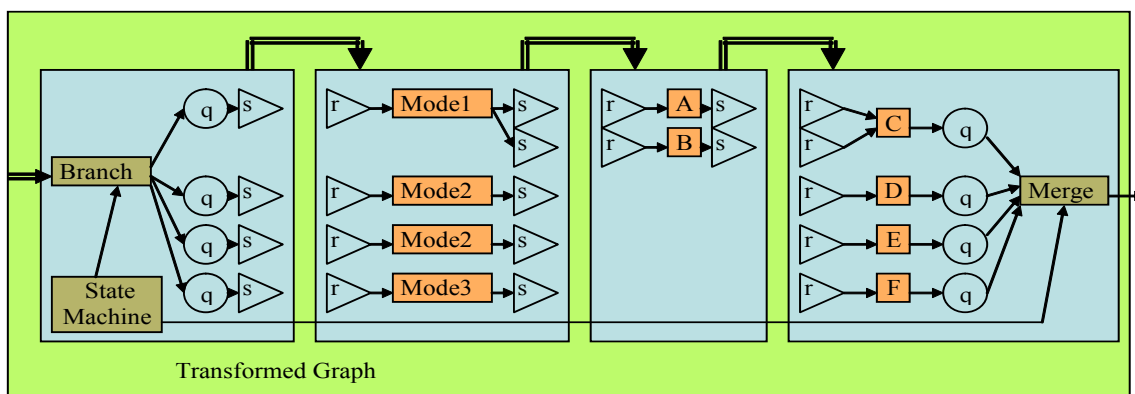
The transformations are the central part of Gedae and make possible the efficient execution of the application expressed in the Gedae Language on the Gedae Virtual Machine. The transformations are fully automated but can be guided by user supplied implementation parameters to control distribution, strip mining, data transfers, scheduling priorities (both static and dynamic), queue policies and memory management. Some of the transformations directly modify the graph into an equivalent graph to implement a user entered implementation decision. For the user to distribute a graph, the user specifies a partitioning of the graph and a mapping of the graph to individual processors. Gedae modifies the graph by inserting send and receive primitives that run on the separate processors and maintain the data flow and connectivity of the graph. The user does not have to modify the graph to achieve these results.

For example, the following graph has dynamic queues and is distributed to four processors by the user:



It is transformed into a new graph, as seen below, with send and receive boxes inserted to manage communications and dynamic queues also inserted to handle dynamic data flow boundaries. Other transformations include modifying the graph to implement strip mining of vectors, adding primitives to implement delay, and adding primitives to allow communication of the graph to the host program or to other Gedae applications. Data structures are also created to implement segmentation, mode control and distributed state coherency.

A sonar signal processing graph will be used to demonstrate how a graph is transformed into an implementation. It will be shown how the transformations can be used to modify the graph execution without changing the Gedae Language expression of the graph. The resulting implementations will be contrasted with how the same implementations would be achieved using traditional programming techniques.



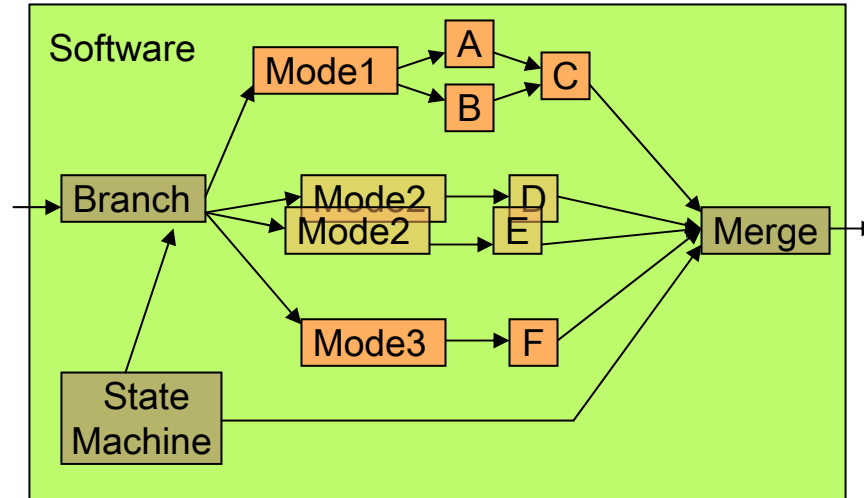
Gedae: Auto Coding to a Virtual Machine

Authors: William I. Lundgren,
Kerry B. Barnes, James W. Steed

What is Gedae?

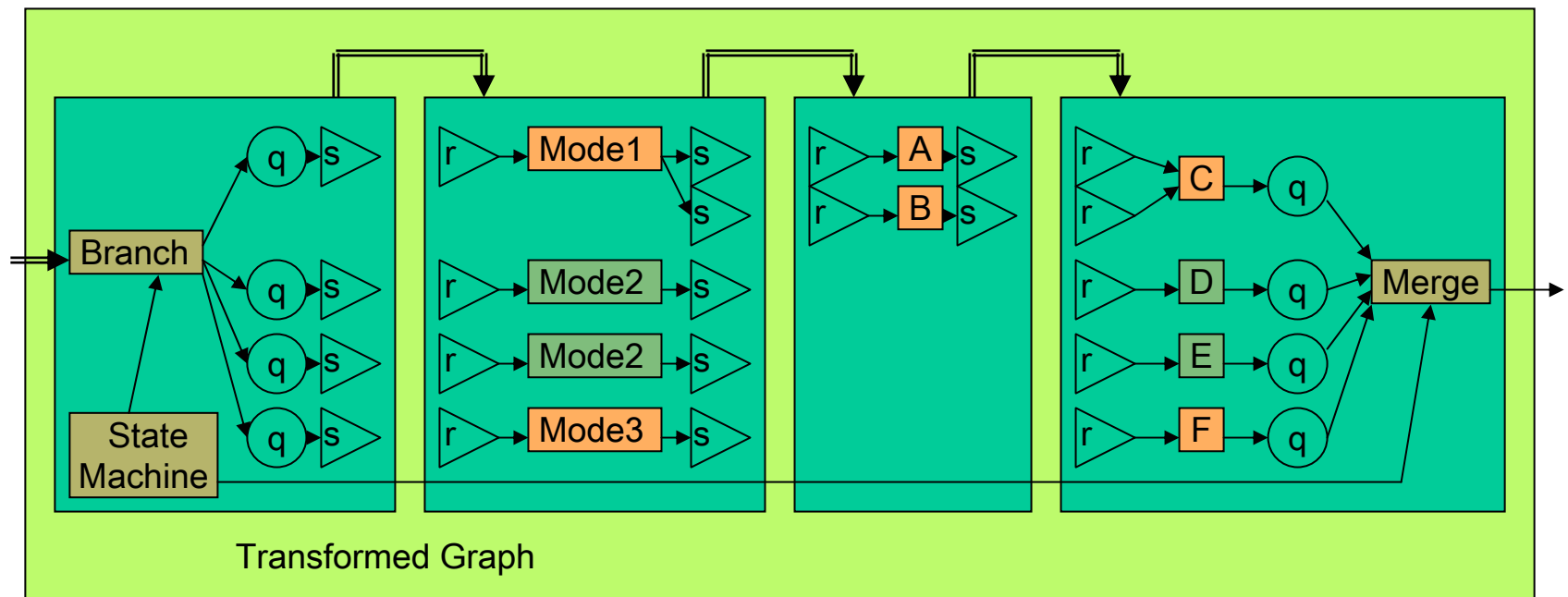


Gedae is a block diagram **language** ...



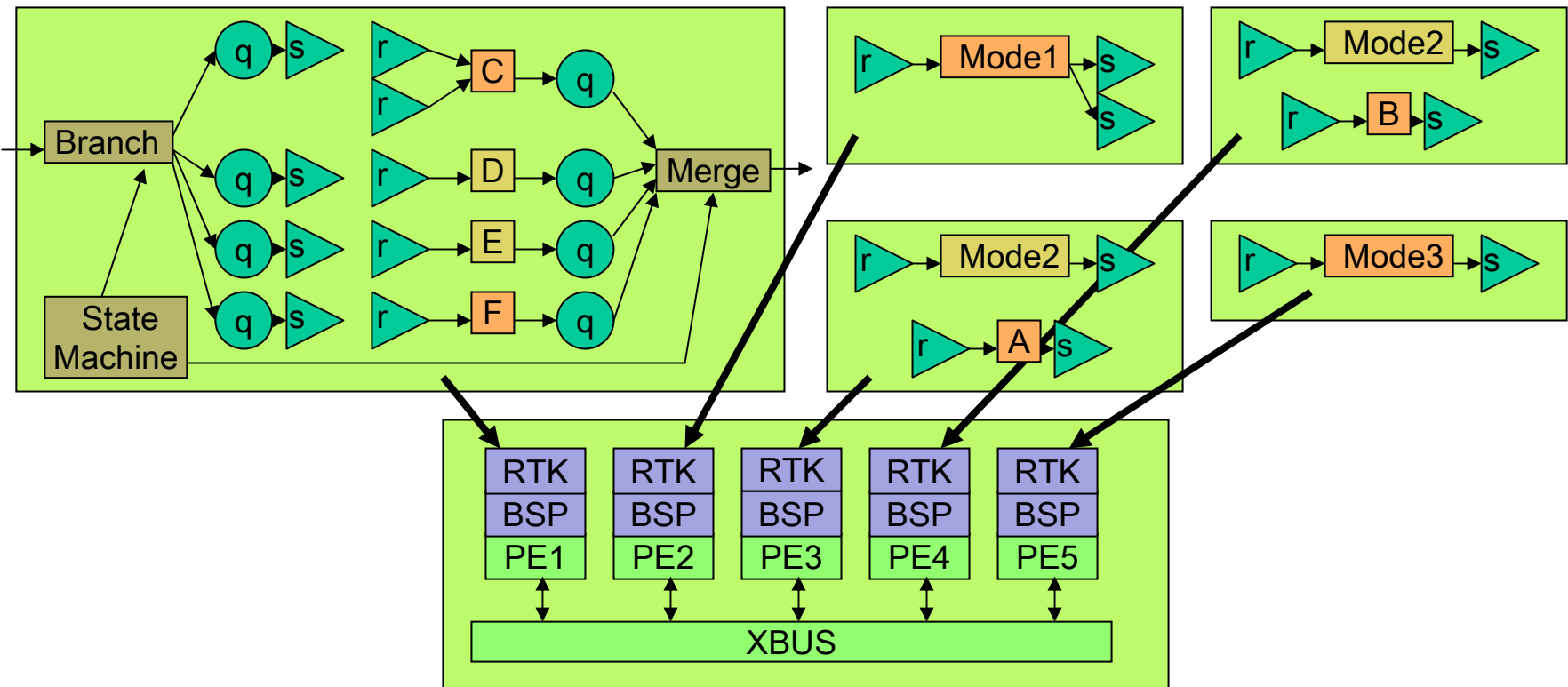
Express signal and data processing algorithms, parallelism, load balancing, fault tolerance and mode control

..that Gedae **transforms** under user control...



User can set optimization parameters that are independent of the graph to guide transformation

...to operate efficiently on a **virtual machine**.

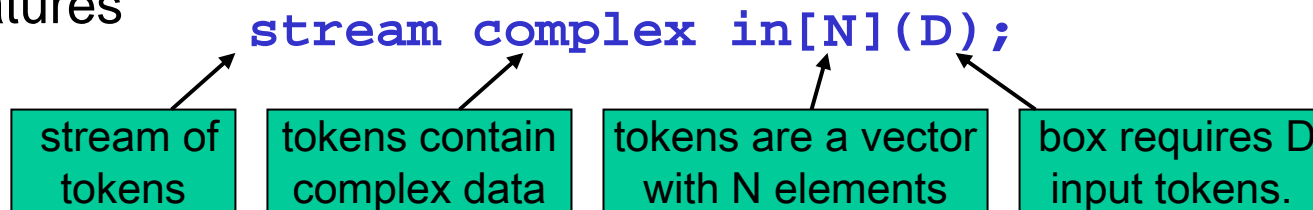
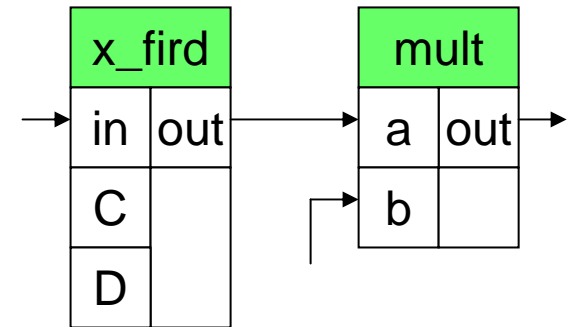


Complete systems can be developed independent of the target system without losing runtime efficiency

Gedae Language



- Gedae provides application information through
 - modules with well-defined behavior
 - ports with well-defined characteristics
 - and manifest connectivity with explicit sequential and parallel execution paths
- This information is implicit in most languages
- Gedae makes the information explicit
 - over 50 different information expression features

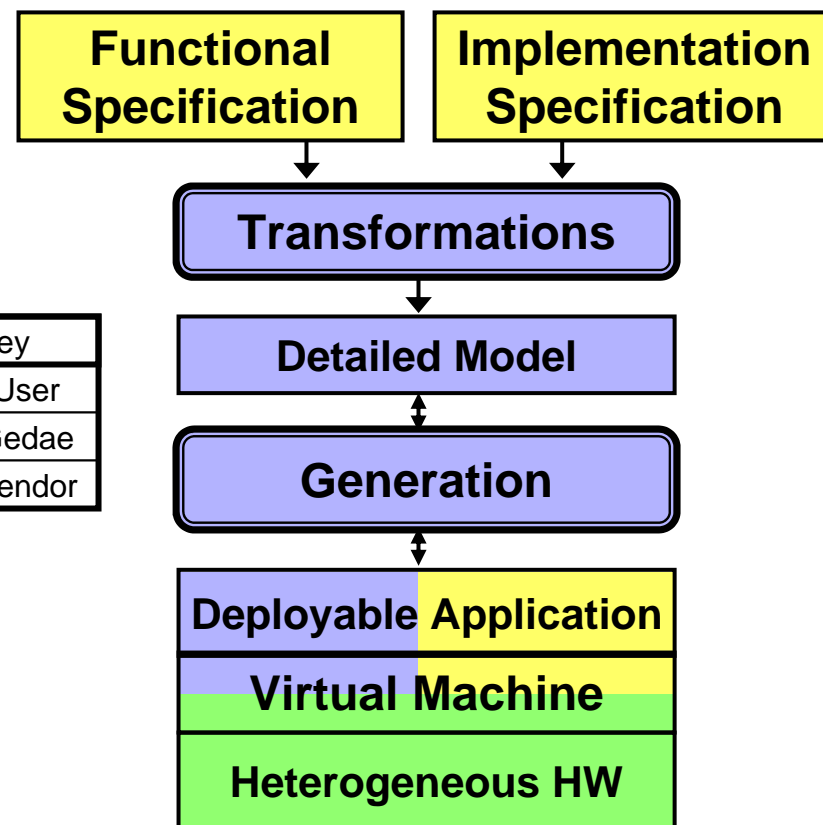


Information provided by language allows Gedae to analyze and efficiently implement algorithms

Gedae Transformations

- The block diagram is transformed using over 100 algorithms.
- The transformations establish the:
 - Order of execution
 - Queue sizes
 - Granularities
 - Memory layout
 - Dynamic schedule parameters
 - Data transfer types and parameters
 - Mode control

Key	
	User
	Gedae
	Vendor

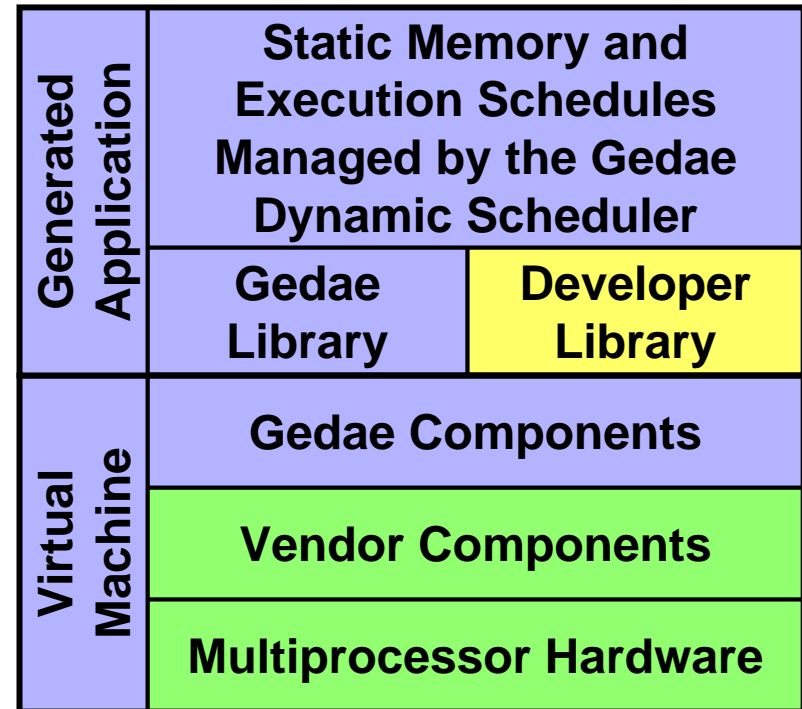


The Gedae transformations build a detailed model of the deployed application. Gedae uses that information to provide visibility

Gedae Virtual Machine (VM)



- Gedae provides the following components:
 - Command handler
 - Dynamic scheduler
 - Segmentation Support
 - Primitive Support
 - Visibility Support
- The vendor provides
 - Inter-processor communications
 - Optimized vector libraries
 - Other basic services



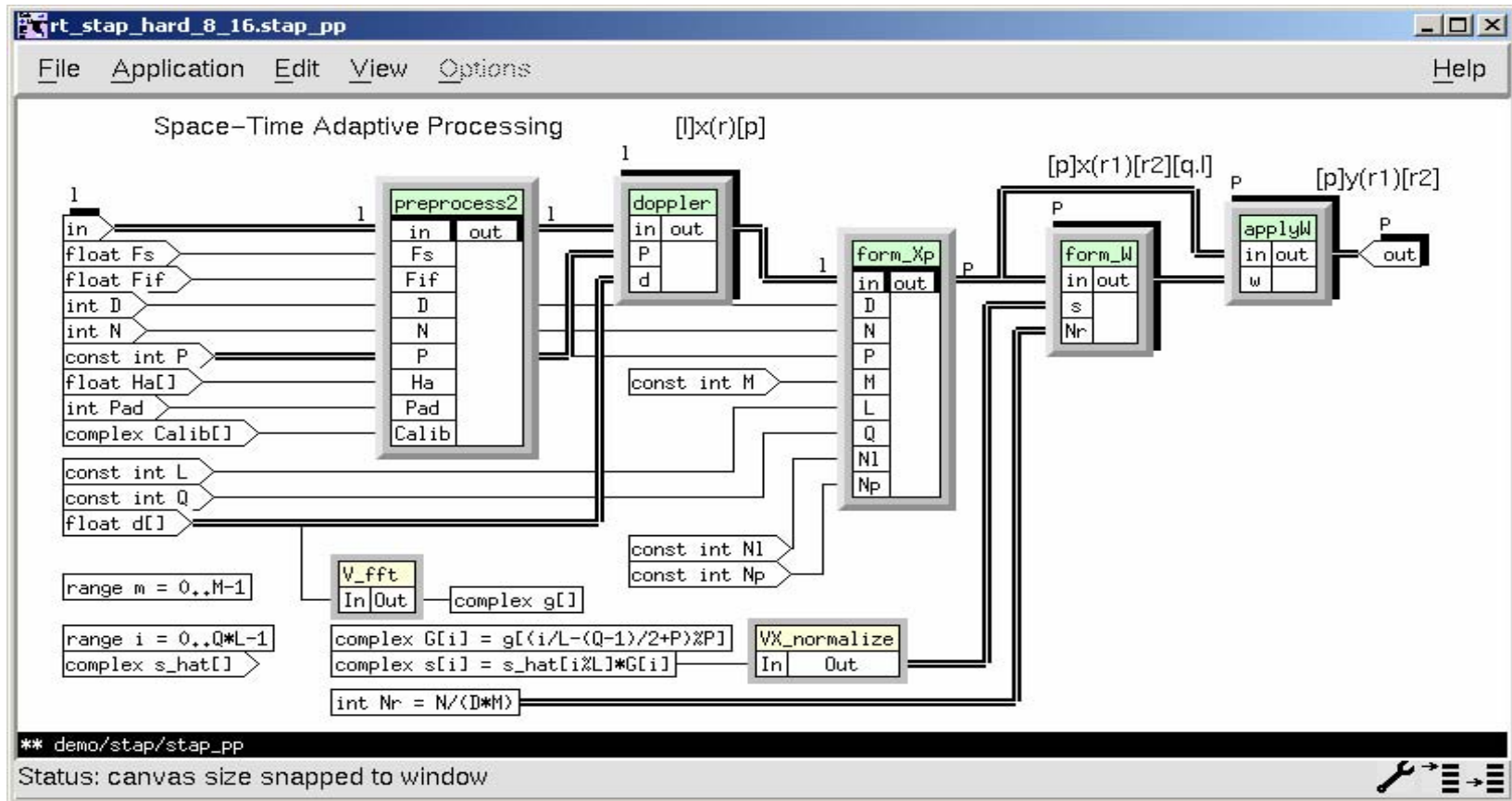
The Gedae virtual machine makes applications processor independent

Three Examples

- Real-Time Space-Time Adaptive Processing (RT-STAP)
 - Miter benchmark graph
 - Illustrates efficient parallel execution of large graph
- Multilevel Mode Graph
 - Illustrates nested mode control with distributed state
 - Dynamic data application
- Sonar Graph
 - Illustrates large data reduction during processing

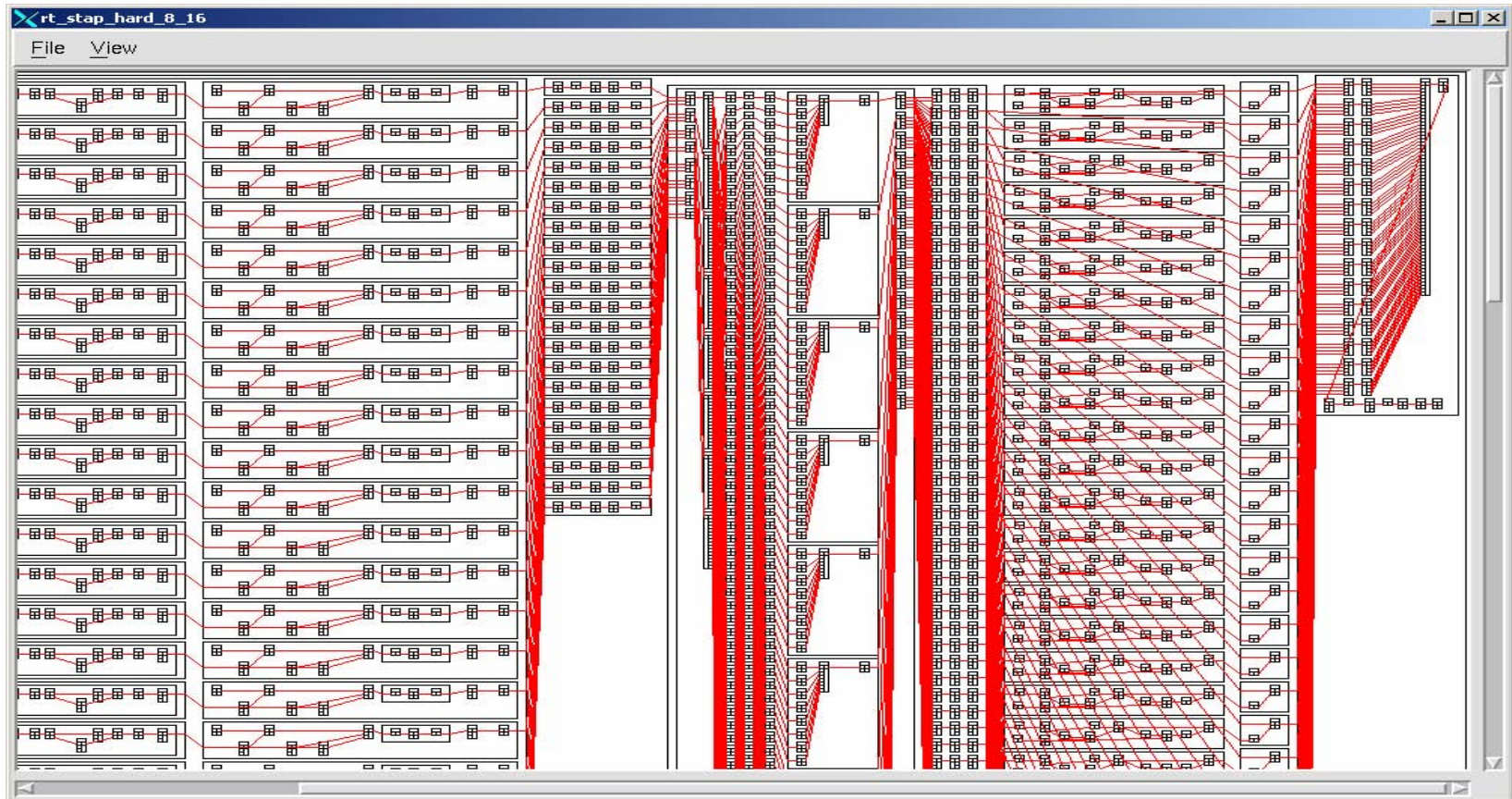
Each example illustrates features of the language, transformations, and virtual machine

RT-STAP: Language



Families permit replicating box and data elements

RT-STAP: Language



- Instantiation constants control the size of the graph
- Routing boxes allow equation based connectivity

RT-STAP: Transformations



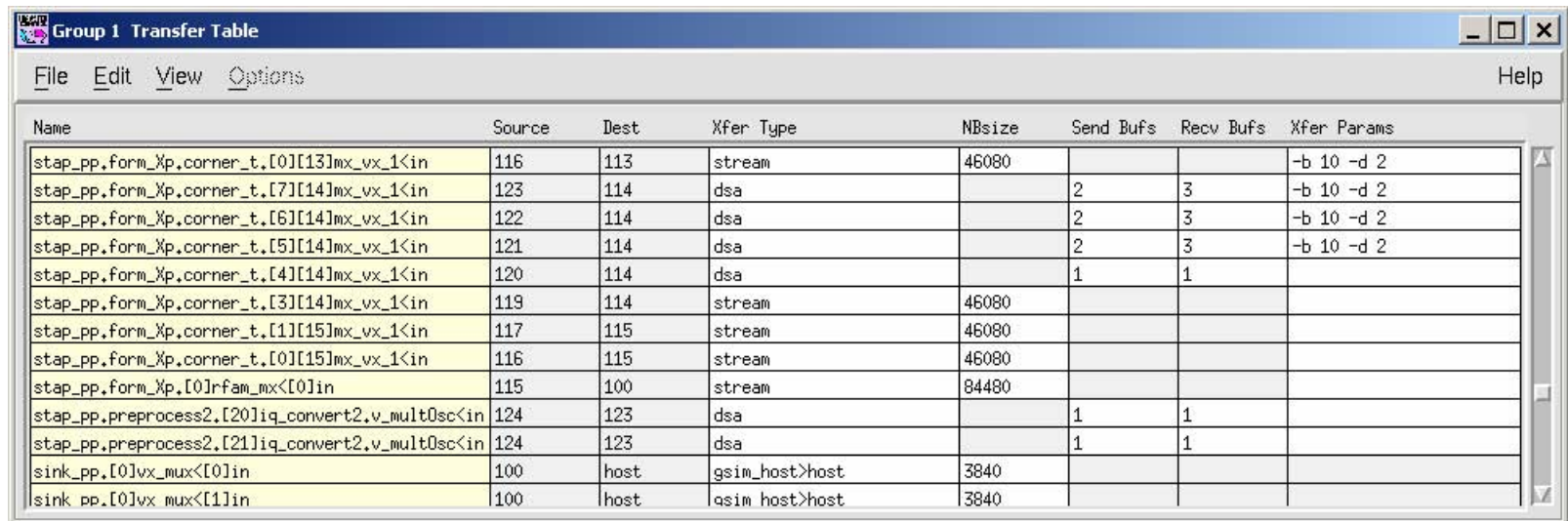
- User maps primitives to physical processors
- Gedae transforms graph by inserting send/receive primitives to communicate between partitions
- Gedae automatically creates executables to run on each processor

Name	Part	SubSched
[55]applyW	113	
[56]applyW	114	
[57]applyW	114	
[58]applyW	114	
[59]applyW	114	
[60]applyW	115	
[61]applyW	115	
[62]applyW	115	
[63]applyW	115	
[0]form_W	100	
[1]form_W	100	
[2]form_W	100	
[3]form_W	100	
[4]form_W	101	
[5]form_W	101	
[6]form_W	101	

Different mappings can be tried without modifying the graph – the needed transformation happens automatically

RT-STAP Transformations

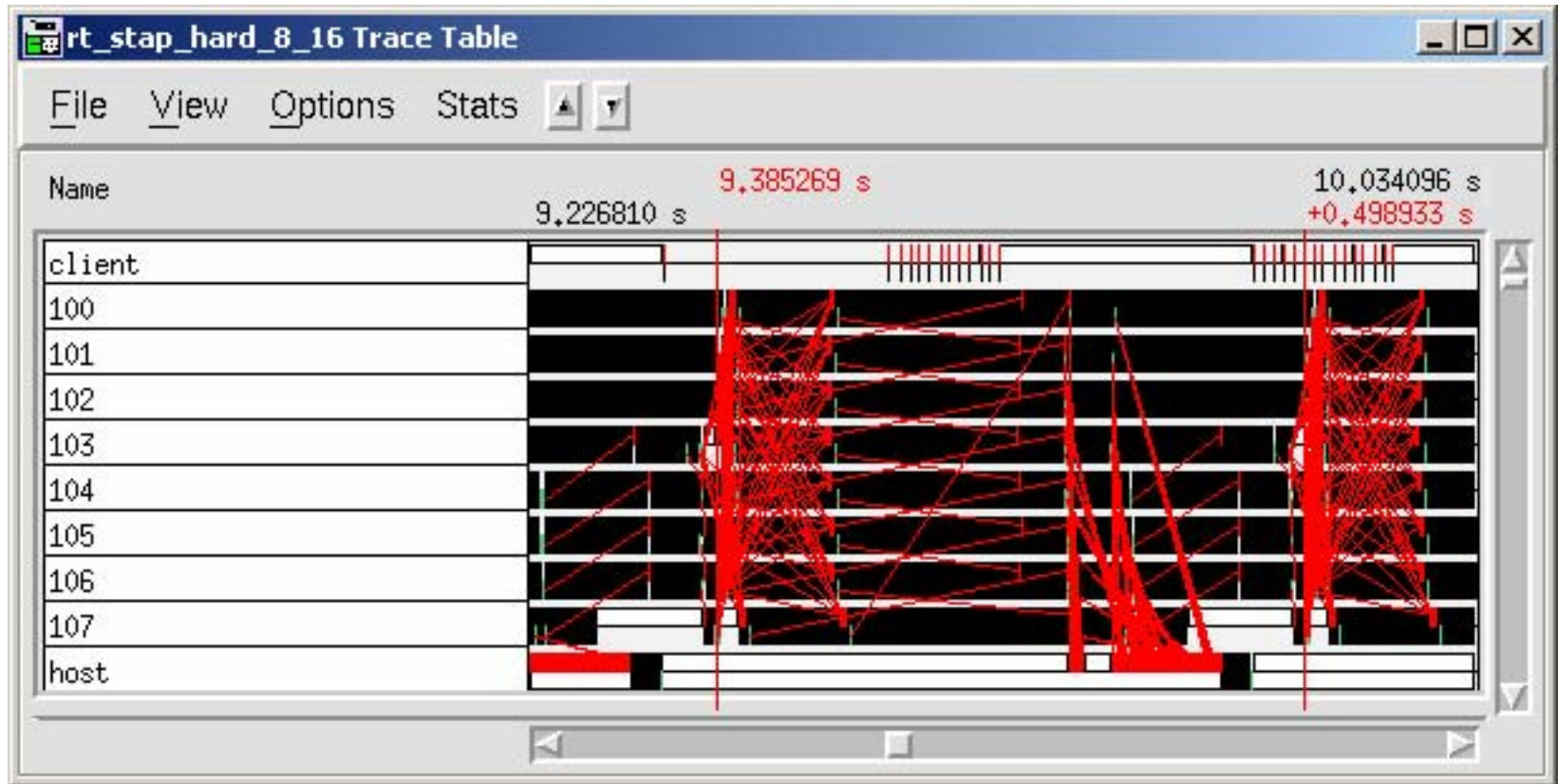
- User can set transfer properties on send/recv pairs with Transfer Table
- Transformations automatically set parameters to send/recv pairs to communicate these properties to running application



Name	Source	Dest	Xfer Type	NBsize	Send Bufs	Recv Bufs	Xfer Params
stap_pp.form_Xp.corner_t.[0][13]mx_vx_1<in	116	113	stream	46080			-b 10 -d 2
stap_pp.form_Xp.corner_t.[7][14]mx_vx_1<in	123	114	dsa		2	3	-b 10 -d 2
stap_pp.form_Xp.corner_t.[6][14]mx_vx_1<in	122	114	dsa		2	3	-b 10 -d 2
stap_pp.form_Xp.corner_t.[5][14]mx_vx_1<in	121	114	dsa		2	3	-b 10 -d 2
stap_pp.form_Xp.corner_t.[4][14]mx_vx_1<in	120	114	dsa		1	1	
stap_pp.form_Xp.corner_t.[3][14]mx_vx_1<in	119	114	stream	46080			
stap_pp.form_Xp.corner_t.[1][15]mx_vx_1<in	117	115	stream	46080			
stap_pp.form_Xp.corner_t.[0][15]mx_vx_1<in	116	115	stream	46080			
stap_pp.form_Xp.[0]rfam_mux<[0]in	115	100	stream	84480			
stap_pp.preprocess2.[20]iq_convert2.v_mult0sc<in	124	123	dsa		1	1	
stap_pp.preprocess2.[21]iq_convert2.v_mult0sc<in	124	123	dsa		1	1	
sink_pp.[0]vx_mux<[0]in	100	host	gsim_host>host	3840			
sink_pp.[0]vx_mux<[1]in	100	host	gsim_host>host	3840			

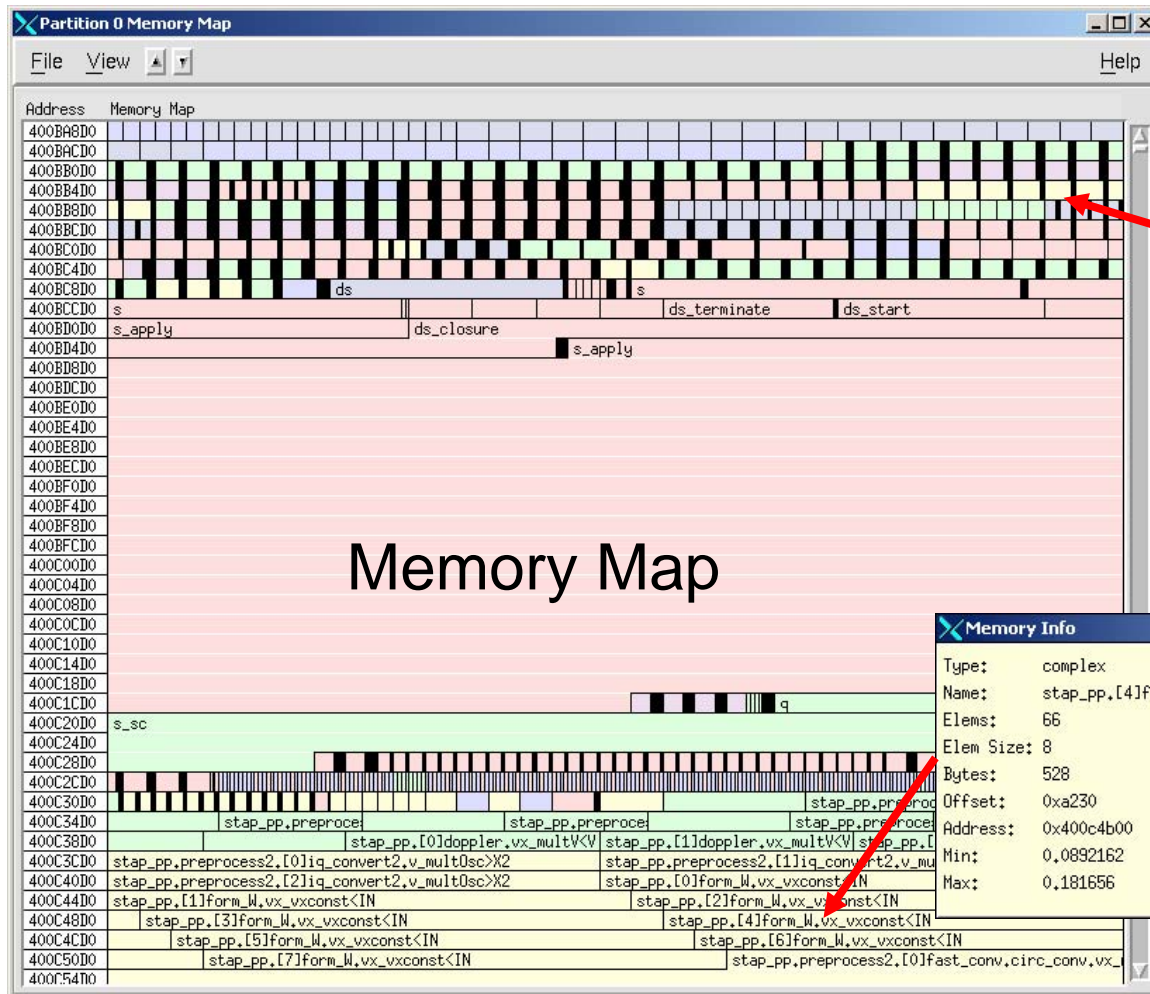
User can guide transformations to optimize implementation

RT-STAP: Running on VM



Send/Recv webs show interprocessor communication and uncover synchronization problems

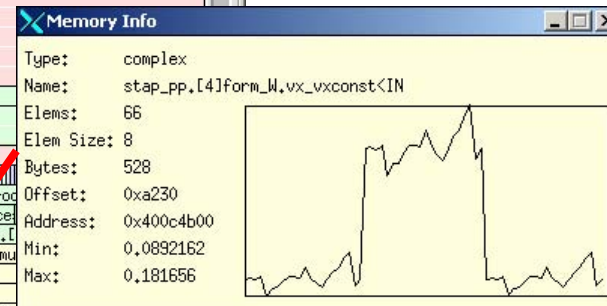
RT-STAP: Running on VM



Memory Map

Name	Value
embeddable/stream/fird state	...
int granularity	31296
float *in	0x4076D610; ZERO_PTR
float *C	0x400C3300; -0.00161023...
int *D	0x400C3008; 4
float *Crvs	0x400C3390; -0.00161023...
int N_Crvs	36
float *out	0x4080FCA0; ZERO_PTR
int N	36

Structure Display



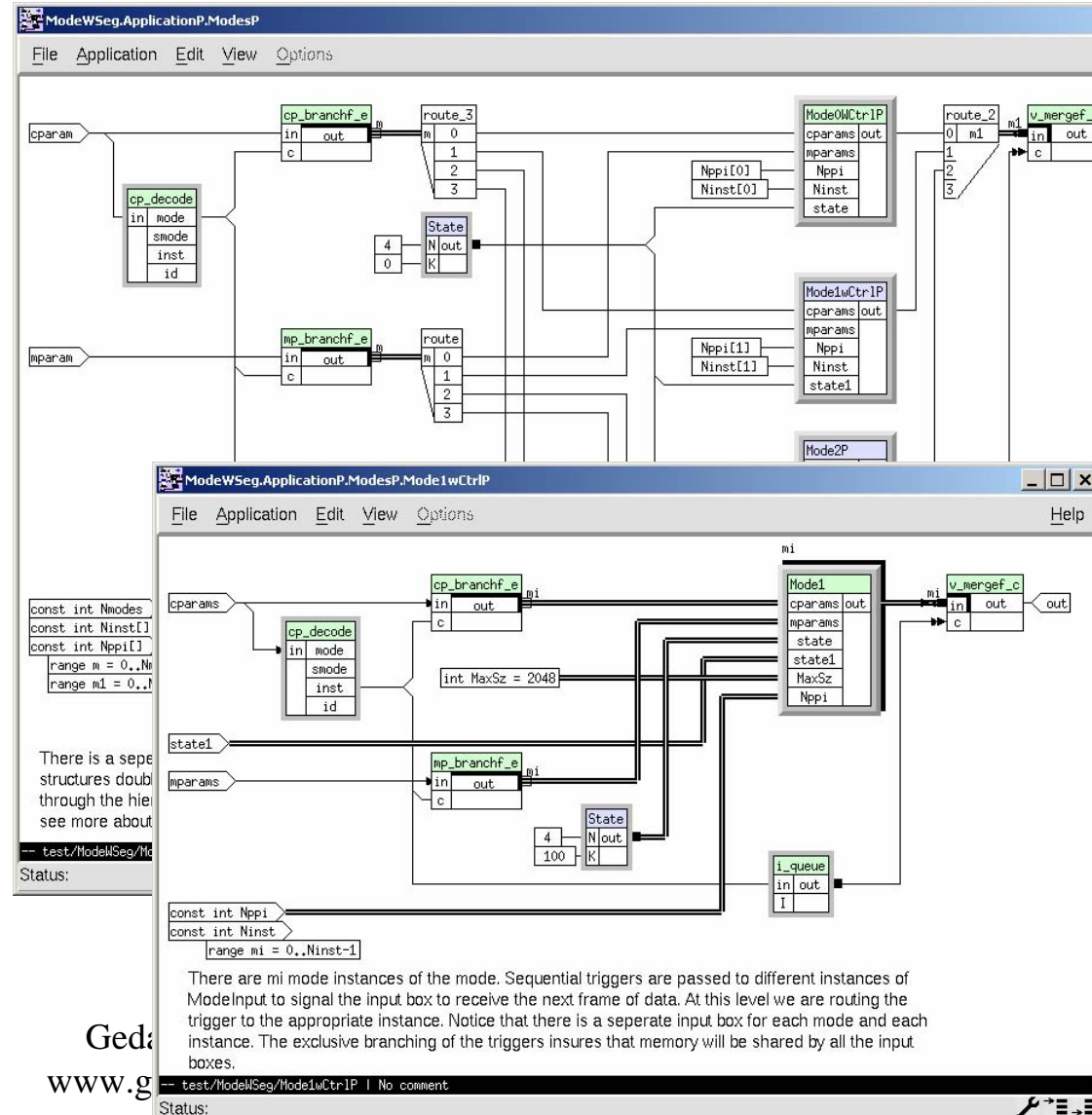
Preplanned use of memory allows distributed runtime debugging

Mode Control: Language



- Branch boxes make mode changes and mark segment boundaries
- “Exclusive” branch outputs show where resources can be shared
- State shared between modes is explicitly declared in the graph

The Gedae primitive language directly supports segmented data processing, sharing of resources, and distribution of state



Mode Control: Language

Branch box copies input data stream to one of a family of outputs based on a control stream. Output is:

- **Segmented** - the box will add **segment** boundaries to the output
- **Dynamic** - the box will state how much data is **produced** on the output at runtime.
- **Exclusive** - only one of the family of F outputs gets data on any firing. Allows sharing of resources and state.

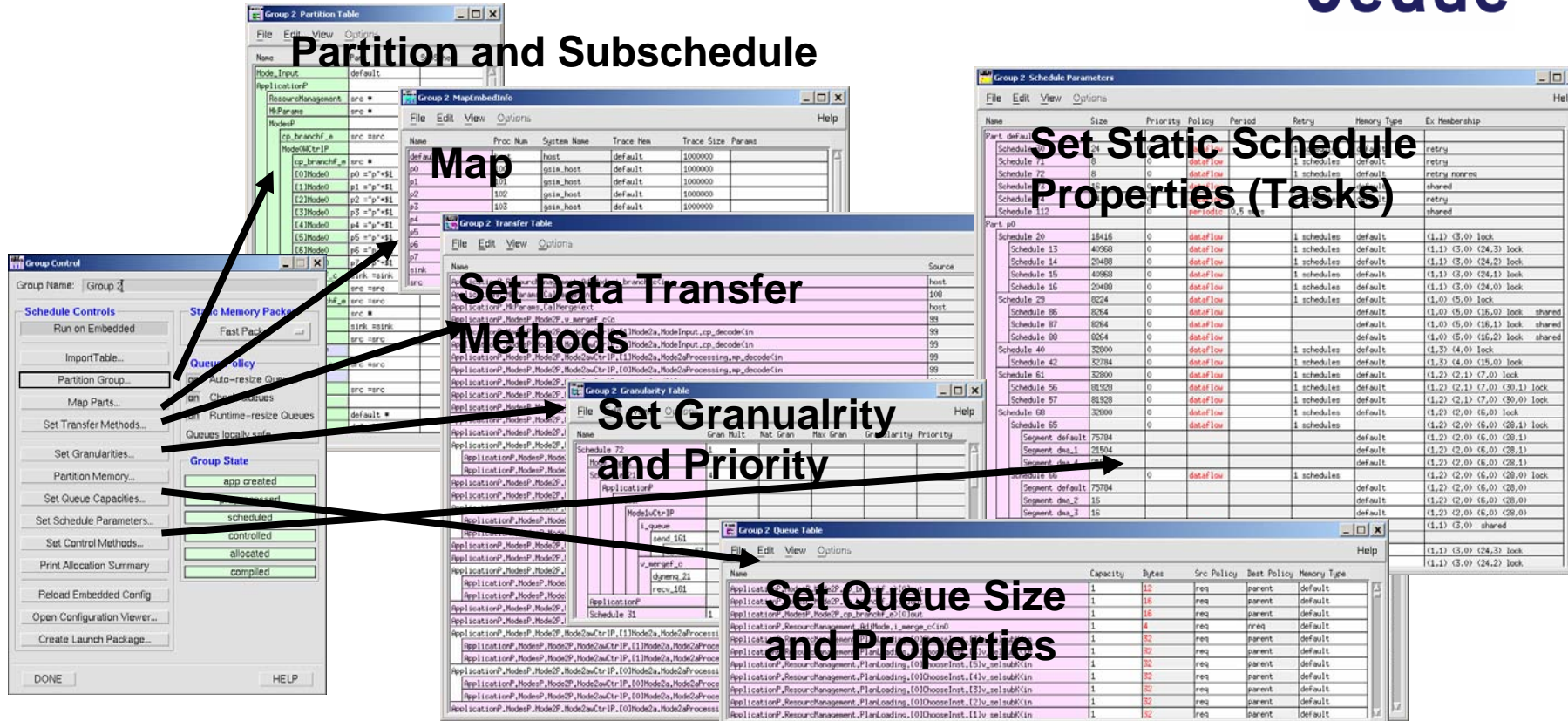
The Gedae extensible language has no “built-in” primitives.
8000+ delivered primitives.
Users can add custom primitives

```

Name: cp_branchf_e
Input:  stream ControlParamRec in;
Input:  stream int c;
Local:  int last;
Output: exclusive segmented dynamic stream
        ControlParamRec [F]out;
Reset: { last = -1; }
Apply: {
    int g,i;
    int prdc = 0;
    for (g=0; g<granularity; g++) {
        int j = c[g];
        if (last != j) {
            if (0<=last && last<F) {
                produce(out[last],prdc);
                prdc = 0;
                segment(out[last],SEGMENT_END);
            }
            last = j;
        }
        if (0<=j && j<F) {
            *out++ = *in;
            prdc++;
        }
        in++;
    }
    produce(out[last],prdc);
}

```


Mode Graph: Transformation



The image displays the Gedae Mode Graph Transformation interface, which includes several dialog boxes for configuring different aspects of the mode graph. Arrows point from text labels to specific dialog boxes:

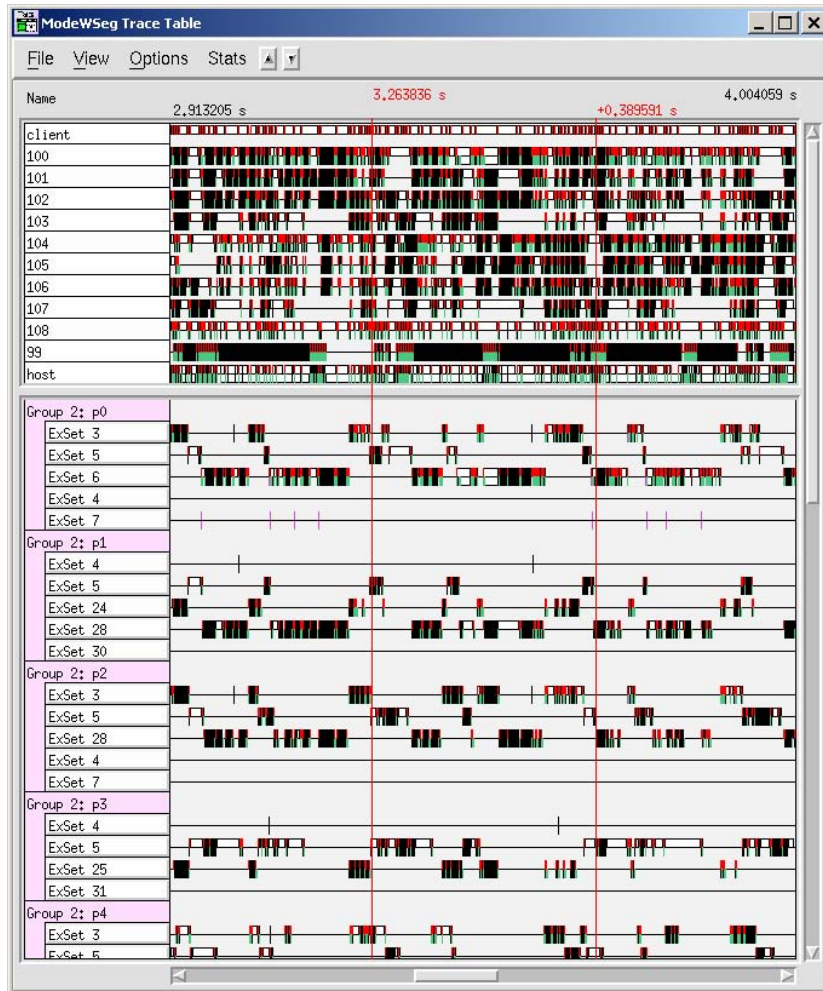
- Partition and Subschedule** points to the **Group 2 Partition Table** dialog.
- Map** points to the **Group 2 MapEmbedInfo** dialog.
- Set Data Transfer Methods** points to the **Group 2 Transfer Table** dialog.
- Set Granularity and Priority** points to the **Group 2 Granularity Table** dialog.
- Set Static Schedule Properties (Tasks)** points to the **Group 2 Schedule Parameters** dialog.
- Set Queue Size and Properties** points to the **Group 2 Queue Table** dialog.

The **Group Control** dialog on the left contains the following options:

- Schedule Controls**
 - Run on Embedded
 - Import Table...
 - Partition Group...
 - Map Parts...
 - Set Transfer Methods...
 - Set Granularities...
 - Partition Memory...
 - Set Queue Capacities...
 - Set Schedule Parameters...
 - Set Control Methods...
 - Print Allocation Summary
 - Reload Embedded Config
 - Open Configuration Viewer...
 - Create Launch Package...
- Static Memory Packets**
 - Fast Pack...
- Queue Policy**
 - Auto-resize Queue
 - Runtime-resize Queues
 - Queues locally safe
- Group State**
 - app created
 - scheduled
 - controlled
 - allocated
 - compiled

User can set partitioning, mapping, data transfer methods, granularity, priority, queue sizes and schedule properties from the group control dialog

Mode Graph: Running on VM



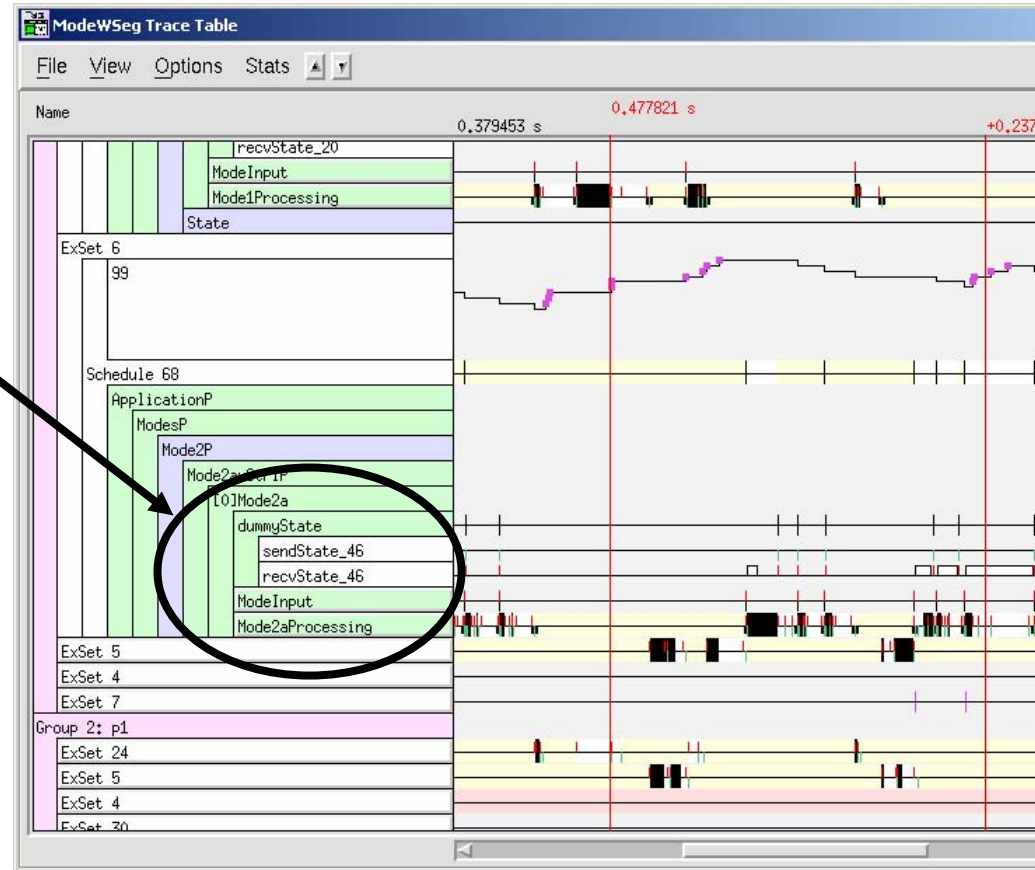
- Each mode requires a different number of processors
- Branch boxes at one level are responsible for the dynamic distribution

VM runtime kernel enforces dynamic data driven execution. Send and receive primitives and state transfer primitives use BSP of virtual machine to transfer data

Mode Graph: Running on VM

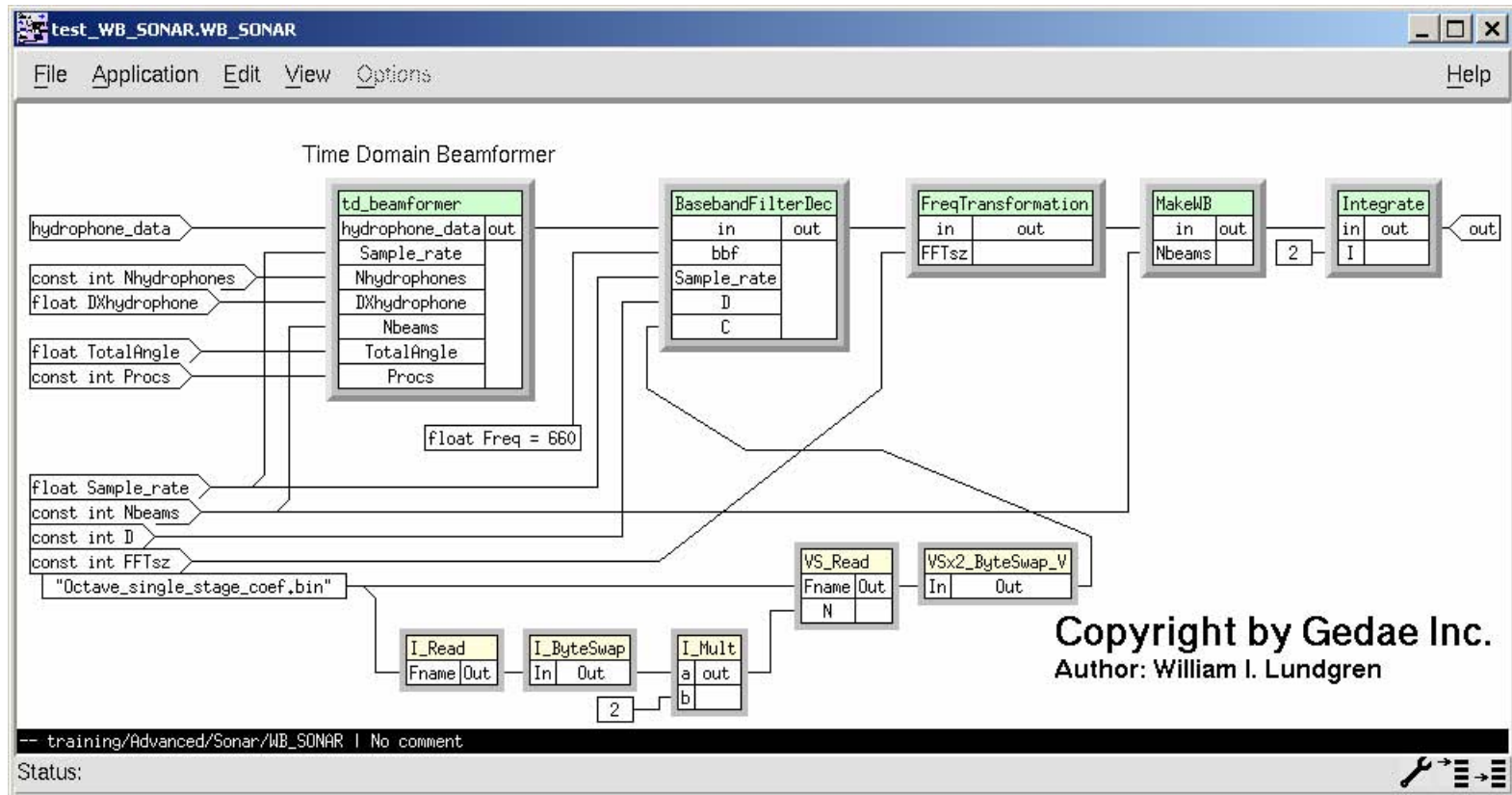


- Primitives to send and receive state are automatically added by transformations
- Messages generated by Virtual Machine at mode change boundaries efficiently coordinate state transfers



Result is efficient transparent use of shared state on distributed processing system

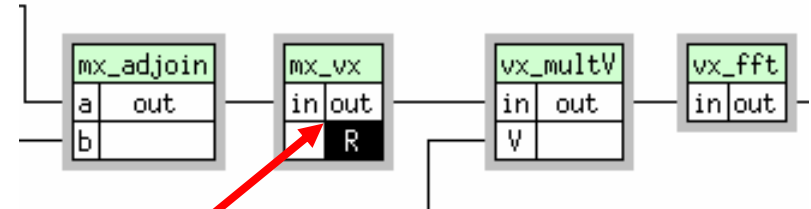
Sonar: Language



Sonar Graph creates low bandwidth output from high bandwidth input data

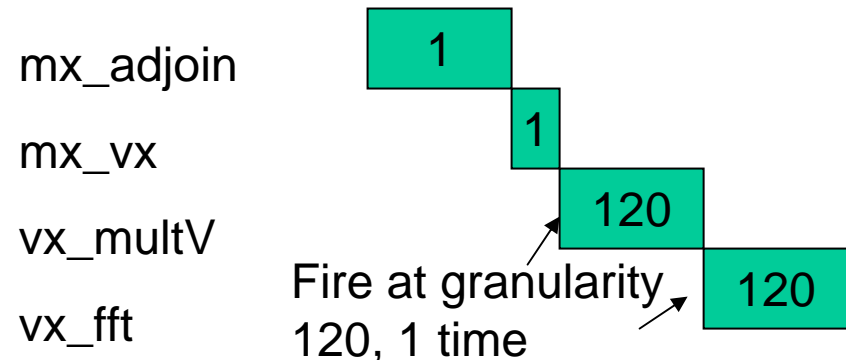
Sonar: Language

- Connectivity + Port Descriptions gives information needed to schedule graph
- `mx_vx` produces $R=120$ tokens out for every 1 token in
- `vx_multV` box must fire 120 times for each firing of the `mx_vx` box.
- `vx_fft` box fires one time for each firing of `vx_multV` box
- Simple predetermined schedule generated from graph and info embedded in primitives



inplace stream complex $\text{out}[C](R) = \text{in};$

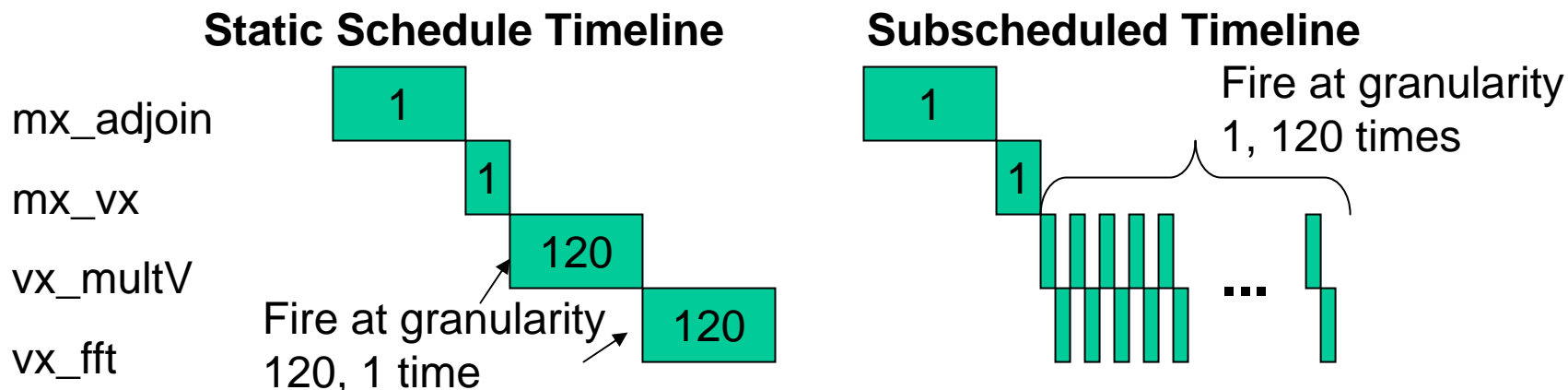
Static Schedule Timeline



Can create a multirate graph that has boxes firing at different granularities

Sonar: Transformation

- User can place boxes in subschedules to strip-mine the vector processing
- Allows use of fast memory
- Can reduce memory usage



Multirate graphs can be implemented using subscheduling to improve speed and reduce memory usage

Sonar: Transformation

Auto-Subscheduling Tool

Group 1 Gain Hier Table

File Edit View Options Help

Name	TotalG	Bytes	Div	TDiv	G	Bytes*G	Boxes	Subsched
Schedule 1	1	1440	1	1	1	1440	6	
1	2	1966080	2	2	1	1966080	2	1
2	240	20484	30	60	4	81936	6	1.1
3	1024	4800	32	64	16	76800	1	1.2
4	4096	63628	64	4096	1	63628	17	1.2.1

Schedule Information Dialog

- User can put boxes into named subschedules manually – but can be difficult
- Auto-Subscheduling Tool puts boxes in subschedules automatically
- Finds nested sets of connected boxes running at common granularities.
- Automatically sets subscheduling levels

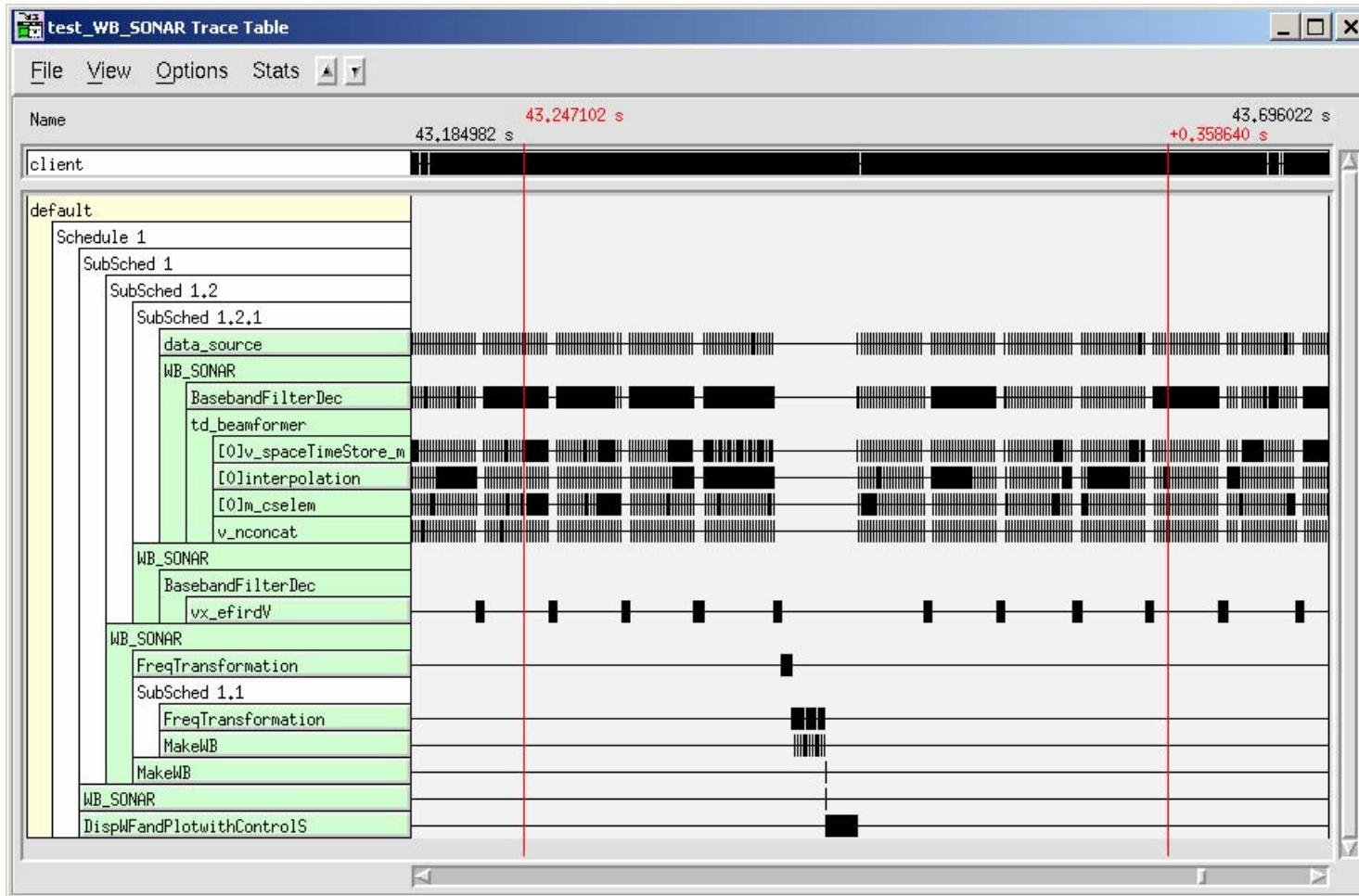
Group 1 Schedule Parameters

File Edit View Options

Name	Size	Priority	Policy	Period
Part default				
Schedule 1	1448	0	dataflow	
SubSched 1				
Segment default	2457600			
Segment parent_memory	480			
SubSched 1.1				
Segment default	65536			
Segment parent_memory	32784			
SubSched 1.2				
Segment default	143040			
Segment parent_memory	15360			
SubSched 1.2.1				
Segment default	61720			
Segment parent_memory	960			

Auto-subscheduling has reduced memory needed by graph from 250 Mbytes to about 2.5 Mbytes - 100x improvement

Sonar: Running on VM



Multiple levels of subscheduling evident on Trace Table

Conclusion

- Gedae Block Diagram Language allows simple expression of a wide range of algorithms
- User optimization information can be added without modifying block diagram
- 100+ transformations create efficient executable application from language and user information
- Application runs efficiently on Virtual Machine
- VM provides portability and visibility

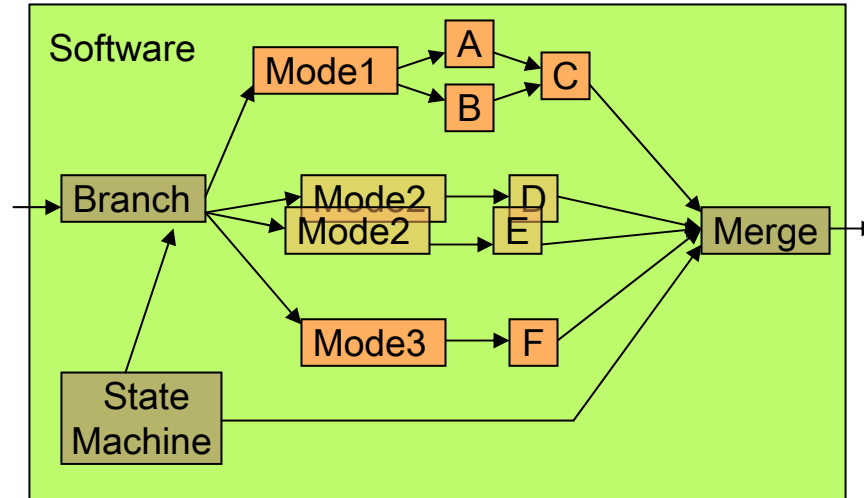
Gedae: Auto Coding to a Virtual Machine

Authors: William I. Lundgren,
Kerry B. Barnes, James W. Steed

What is Gedae?



Gedae is a block diagram **language** ...

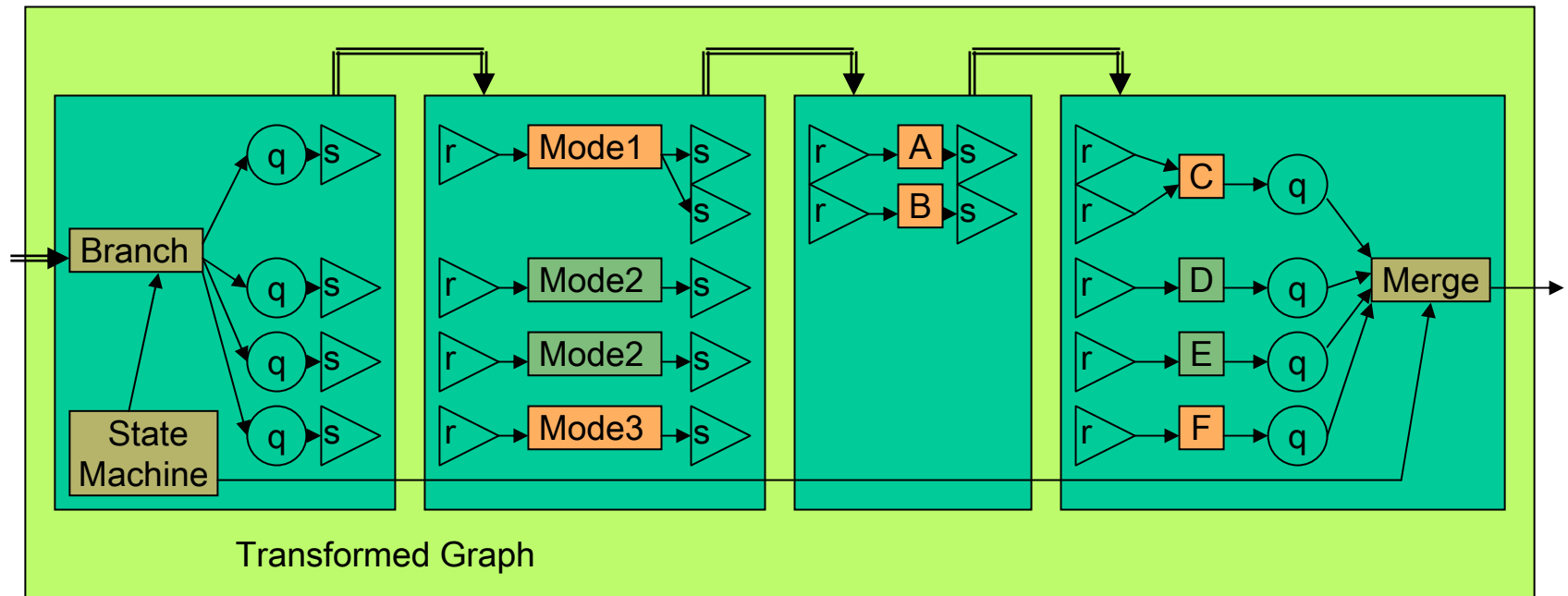


Express signal and data processing algorithms, parallelism, load balancing, fault tolerance and mode control.

What is Gedae?



..that Gedae **transforms** under user control...

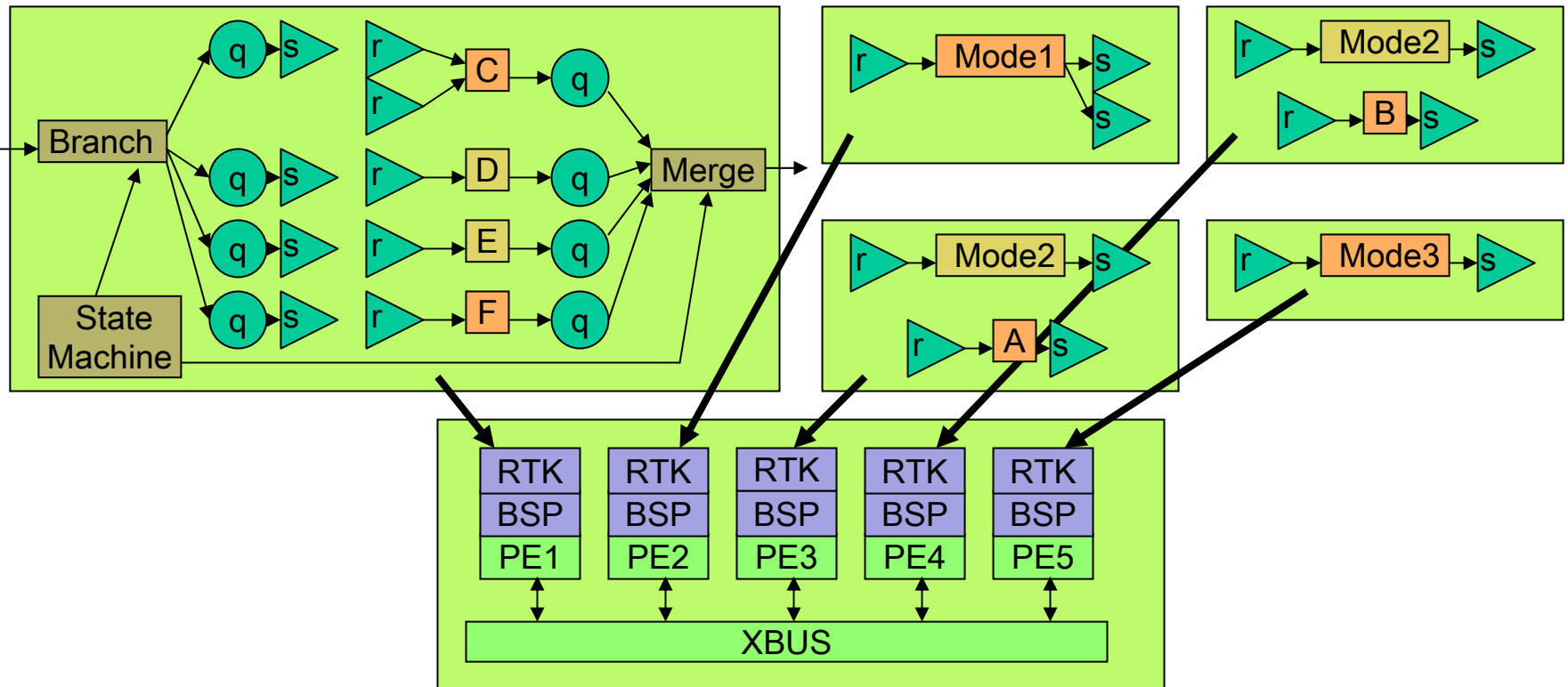


User can set optimization parameters that are independent of the graph to guide transformation

What is Gedae?



...to operate efficiently on a **virtual machine**.

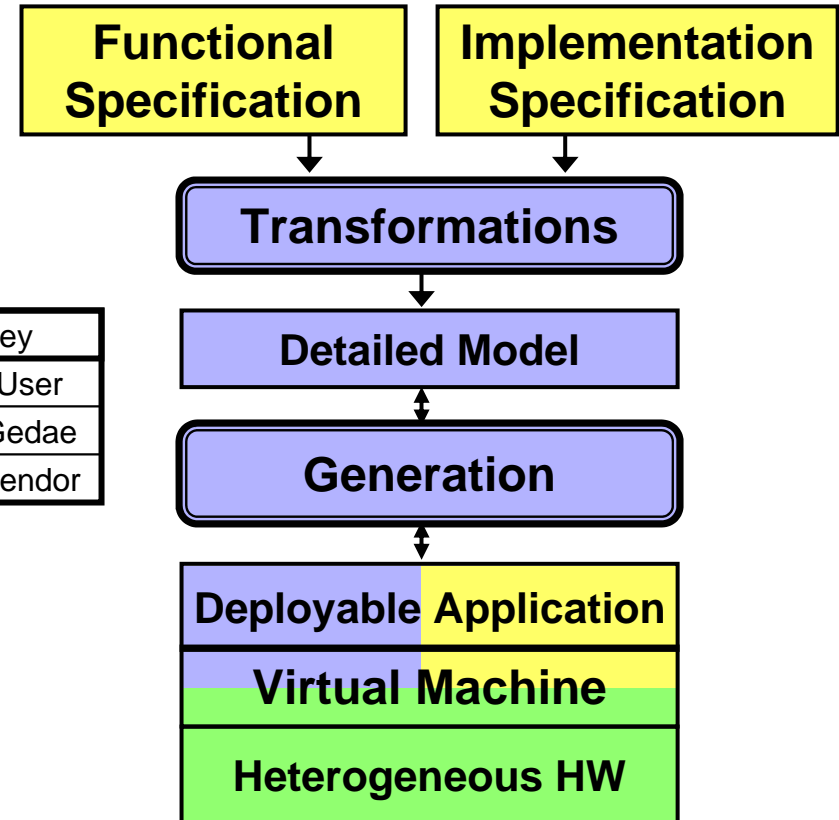


Complete systems can be developed independent of the target system without losing runtime efficiency.

Gedae's Structure

- The block diagram is transformed using over 100 algorithms.
- The transformations establish the:
 - Order of execution
 - Queue sizes
 - Granularities
 - Memory layout
 - Dynamic schedule parameters
 - Data transfer types and parameters
 - Mode control

Key	
	User
	Gedae
	Vendor



The Gedae transformations build a detailed model of the deployed application. Gedae uses that information to provide visibility.